Notes diverses (trouvé sur Internet)

- Tracer l'exécution d'un script perl

```
perl -d:Trace myprog.pl
```

- UTF8

Opérateur diamant en UTF8

```
# La lecture de <> se fera en UTF8, et un fichier ouvert sera par défaut
# également en UTF8.
use open qw(:utf8);
```

Entrées/sorties en UTF8

```
# Les entrées/sorties standard (ainsi que l'erreur) se feront en UTF8
binmode $_, ':utf8' for \*STDIN, \*STDOUT, \*STDERR;
```

Pour faire les deux ci-dessus simultanément

```
# Passe <> (ainsi que tout fichier ouvert par défaut), STDINT, STDOUT et STDERR
# en utf8.
use open qw (:std :utf8);
```

Notes du livre online *Modern Perl*

####Auteur : chromatic ####Éditeur : Michael Swaine

####Télécharger le document

http://onyxneon.com/books/modern_perl/modern_perl_2016_a4.pdf

---

- `weaken`

Désactiver le comptage des références

- `wantarray`

Détecter le contexte d'appel

- `each`

Parcourir les éléments d'un tableau ou hash par indice / valeur (tableau) ou clé / valeur (hash)

- `keys`

Parcourir les clés d'un hash

- `values`

Parcourir les valeurs d'un hash

- Supprimer les valeurs en double dans un tableau

```
my %uniq;
undef @uniq{@tab};
my @tab_uniq = keys %uniq;
```

À noter que cette méthode ne conserve pas l'ordre.

Version qui conserve l'ordre :

```
# D'après https://www.perlmonks.org/?node_id=280714
my %seen;
@uniq = grep ! $seen{$_}++, @list;
```

La bonne méthode :

```
use List::Util qw(uniq);

my @subset = uniq @values;
```

Voir aussi `uniqnum` et `uniqstr`.

- `//=`

Assigner seulement si lvalue est `undef`, exemple :

```
$v //= '<no string provided>';
```

- `//`

Vérifie si le terme à gauche est défini, exemple :

```
sub f { my $str = shift // '<no str provided!>'; }
```

- `lock_keys`

Verrouille les clés d'un hash (`Hash::Util`)

- `lock_value`

Verrouille les valeurs d'un hash (`Hash::Util`)

- `lock_hash`

Verrouille le hash dans sa totalité (`Hash::Util`)

- Pour définir des dualvars

```
Scalar::Util 'dualvar';
```

Notes du livre online *Higher Order Perls*

####Auteur : Mark Jason Dominus ####Éditeur : Elsevier

####Télécharger le document

http://hop.perl.plover.com/book/pdf/HigherOrderPerl.pdf

- En perl, false = error et true = succès

(The hard way)

- `Params::Validate`

Validate method/function parameters

- Option `-MCarp::Always`

Affiche la pile d'appel en cas d'erreur.

- `IPC::Open3`

"Open a process for reading, writing, and error handling using open3()"

- `Devel::NYTProf` (ou bien `Devel::DProf`, mais le premier est mieux)

Profiling performance

- `Perl::Critic`

Analyzing code quality. It's basically lint for Perl.

- `Data::Dumper`

Analyzing a variable's structure

- `Params::Validate` (ou bien, pour Moose : `MooseX::Method::Signatures`)

Vérification de paramètre à l'appel de fonction.

- `Contextual::Return`

Retourne une valeur en fonction du contexte d'appel.

- Code trouvé ici et là

```
my @files = grep { -f } glob( "*" );

my $casefix = sub { return ucfirst lc $_[0] };

use warnings FATAL => 'all';

my $s = eval `' sub { $text } `' ;
```

```
use DB_File;
tie %hash => 'DB_File', $file, ...

use Term::ANSIColor;
```

Notes du livre *Perl One-Liners (130 Programs That Get Things Done)*

####Auteur : Peteris Krumins ####Éditeur : No Starch Press

####Télécharger le code des one-liners (perl1line.txt)

http://www.catonmat.net/download/perl1line.txt

```
"@{[ ... ]}" # Pour exécuter un code dans une chaîne de caractères

perl -le 'print map { ("a".."z")[rand 26] } 1..8'# Mot de passe
perl -le 'print unpack("N", 127.0.0.1)'          # Convertit IP en entier
perl -MSocket -le 'print unpack("N", inet_aton("127.0.0.1"))'# Sans vstring
perl -MSocket -le 'print inet_ntoa(pack("N", 2130706433))'   # Dans l'autre sens
perl -nE 'say' file                              # Ajoute lignes vides
perl -00pe0                                      # Supp. lignes vides multiples
perl -ne '}{print $.'                            # wc -l
    # Détermine si un entier est un nombre premier
perl -lne '(1x$_) !~ /^1?$|^(11+?)\1+$/ && print "$_ is prime"'
perl -Mbignum=bpi -le 'print bpi(21)'            # Affiche 21 décimales de PI
perl -le 'print scalar localtime'                # Affiche date et heure
perl -le 'print join ":", (localtime)[2,1,0]'    # Affiche l'heure locale
    # Calcule factorielle
perl -MMath::BigInt -le 'print Math::BigInt->new(5)->bfac()'
    # Permutations !
perl -MAlgorithm::Permute -le '
@l = (1,2,3,4,5);
Algorithm::Permute::permute { print "@l" } @l
'
use List::PowerSet                               # Partitions d'un ensemble
perl -lpe 'y/A-Za-z/N-ZA-Mn-za-m/' file          # ROT13
perl -MMIME::Base64 -0777 -ne 'print encode_base64($_)' file  # Encode en BASE64
perl -MMIME::Base64 -0777 -ne 'print decode_base64($_)' file  # Decode de BASE64
perl -MURI::Escape -le 'print uri_escape("http://example.fr")'# URL-escape
perl -MURI::Escape -le 'print uri_unescape("http://example.com")' # Inverse...
```

```
perl -pe 's|\012|\015\012|'                          # Sauts de ligne UNIX -> DOS
perl -pe 's|\015\012|\012|'                          # Sauts de ligne DOS -> UNIX
    # Adresse IP
my $ip_part = qr/[0-9]|[0-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5]/;
if ($ip =~ /^$ip_part\.$ip_part\.$ip_part\.$ip_part$/) {
    print "valid ip\n";
}
    # ou bien
if ($ip =~ /^($ip_part\.){3}$ip_part$/) { ... }
perl -MEmail::Valid -ne 'print if Email::Valid->address($_)' # Valide adr. mail
perl -MRegexp::Common -ne 'print if /$RE{num}{real}/'  # Trouve un réel
$str =~ s/(\d+)/$1+1/ge                              # Incrémente les entiers
perl -MRegexp::Common -pe 's/($RE{num}{real})/$1+1/ge' # Incrémente les réels
my @nums = $text =~ /\d+/g;                          # Extrais les nombres...
perl -alne 'print "@{[grep { $_ eq reverse $_ } @F]}"' # Affiche les palindromes
    # Lit par paquets de 1024 octets :
$/ = \1024

perl -le 'print "$_: $ENV{$_}" for keys %ENV'      # Affiche l'environnement
```

Notes du livre *Intermediate Perl, 2nd edition*

####Auteur : Randal L. Schwarz, brian d foy & Tom Phoenix ####Éditeur : O'Reilly

Modules et versions

- Afficher quelle version de module est venue avec une certaine version de Perl

```
use Module::CoreList;
print $Module::CoreList::version{5.01400}{CPAN};
```

- Afficher quelle version de Perl a intégré un module dans la librairie standard

```
use Module::CoreList;
print Module::CoreList->first_release( 'Module::Build' );
```

- ou bien, depuis la ligne de commande

```
corelist Module::Build
```

- Installer un module dans un répertoire spécifique

```
perl Makefile.PL INSTALL_BASE=/home/machin/myperl
```

- PERL5LIB

Pour ajouter un répertoire où chercher les modules.

- Pour afficher @INC

```
perl -le 'print for @INC'
```

- `local::lib`

Permet d'avoir la listes des variables à définir (en syntaxe *bash*) pour faire fonctionner Perl sans droits d'administration.

```
perl -Mlocal::lib
```

- `cpan`, `cpanp`, `cpanm`
  - `cpan` : le client CPAN historique
  - `cpanp` : un client CPAN enrichi ('CPANPLUS')
  - `cpanm` : un client CPAN simplifié ('CPANMINUS'), qui tente de tout automatiser
- `use` et `require`

```
use List::Util;

# Est équivalent à

BEGIN { # what use is really doing
    require List::Util;
    List::Util->import(...);
}
```

- Module `B::Deparse`

Ce module permet d'afficher le script tel que compilé par Perl.

```
perl -MO=Deparse myprog.pl
```

- Pour vérifier un lien HTTP

```
use HTTP::SimpleLinkChecker qw(check_link)
check_link( $_ );
! HTTP::SimpleLinkChecker::ERROR;
```

- Créer un hash d'après un tableau

```
my %hash = map { $_, 1 } @array;
```

- Éviter de répéter le nom d'un variable

```
my $v = do {
  if ( ... ) { 'value1' }
  elsif ( ... ) { 'value2' }
  else { 'value3' };
```

- Lire l'intégralité d'un fichier

```perl
my $file_content = do {
  local $/;
  local @ARGV = ( $file_name );
  <>
};
```

- Fonctions de manipulation de noms de fichier

```perl
use File::Spec::Functions; # Utilise les fonctions plutôt que le mode objet
```

- Faire un grep en interceptant une mauvaise regex

```perl
print grep { eval { /$regex/ } } glob( ".* *");
```

- Le sigil n'est pas forcément collé au nom de variable

```perl
my $ abc = 'youpi';
print $ abc, "\n";
# Idem avec @ ou %
# Par contre, ne fonctionne pas dans une chaîne interpolée
```

- Dé-référencer sans accolades

```perl
@$myarrayref;
%$myhashref;
$$myarrayref[0];
$$myhashref{mykey};
```

- Pour savoir si une variable donnée est un hash au sens large

```perl
# Ne vérifie pas "ce que c'est" avec un ref, plutôt, vérifie que "ça" peut se
# comporter comme un hash, quoi que ce soit.
sub is_hash_ref {
  my $hash_ref = shift;
  rerurn eval { keys %$hash_ref; 1 };
}
```

- L'auto-vivification fonctionne avec n'importe quel *undef*

```perl
my $not_yet;
@$not_yet = ( 1, 2, 3 );

my $top;
$top->[2]->[4] = 'bla';
```

- Déboguer un script

```
perl -d myscript.pl
```

Ensuite : **s** pour avancer d'un pas, **x** pour afficher une ou des variables. **h** pour de l'aide.

- Alternatives à *Dumper*

```
use Data::Dump qw( dump );

# Ou bien
use Data::Printer;
...
p( %mon_hash );

# print Data::Dumper->Dump( 2 array references,
# one for the content, one for the variable names );
print Data::Dumper->Dump(
  [ \@data1, \@data2 ],
  [ qw(*data1 *data2) ]
);

# --->>>
@data1 = (
  'one',
  'won',
  [
    'two',
    'too',
    'to',
    \@data1
  ]
);
@data2 = @{$data1[2]};
```

- Module *Storable*

```
use Storable;
my $frozen = freeze [ \@data1, \@data2 ];
...
my $data = thaw( $frozen );

nstore [\@data1, \@data2], $filename;
...
my $array_ref = retrieve $filename;

# Create a full copy (whatever recursion level)
my $full_copy = dclone $original;
```

- Autres modules pour sérialiser

```
# YAML
use YAML;
...
print Dump( \%mon_hash );
```

```
# JSON
use JSON;
print to_json( \%mon_hash, { pretty => 1 } );
...
my $hash_ref = from_json( $json_string );
```

- Référence vers la sub actuelle

__SUB__ est une référence vers la routine dans laquelle nous nous trouvons, fonctionne aussi bien avec une sub anonyme que régulière.

- *MagicalCodeRef* (page 110)

Permet d'afficher le nom de la sub en cours ainsi que des informations utiles (les closures en cours notamment).

- Un dumper très puissant, notamment, pour les coderef

```
use Data::Dump::Streamer;

Dump $var # can be or contain coderef
```

- Ouvrir un handle de fichier sur une chaîne

```
open my $fh, '>', \ my $string;

# Autre exemple (rediriger STDOUT vers une chaîne)
{
  local *STDOUT;
  open STDOUT, '>', \ my $string;

  print "hello...\n";

  $some_obj->noisy_method();
}
```

- Module *IO::Handle*

```
# Utilisation d'un handle de fichier en mode objet
# La ligne
#   use IO::Handle;
# est optionnelle sur mon perl...
use IO::Handle;

open my $fh, '>', $filename or die "Error: $!";
$fh->print( "bonjour !\n" );
$fh->close;
```

- Module *IO::File*

```
use IO::File;
```

```
my $read_fh = IO::File->new( 'myfile', 'r' );
my $write_fh = IO::File->new( 'myfile', 'w' );

my $append_fh = IO::File->new( 'myfile', O_WRONLY|O_APPEND );
```

- Module *IO::Scalar*

Pour écrire dans une chaîne, ancienne mode.

- Module IO::Tee

Pour multiplexer.

```
use IO::Tee;

my $tee_fh = IO::Tee->new( $log_fh, $scalar_fh );

# Peut aussi multiplexer entre une entrée (dans ce cas, le premier handle donné
# à IO::Tee) et plusieurs sorties.

my $tee_fh = IO::Tee->new( $read_fh, $log_fh, $scalar_fh );
```

- Module *IO::Pipe*

Permet de récupérer le résultat d'une commande externe (ou bien, envoyer des données vers une commande externe), de manière plus propre qu'avec `open`.

```
open my $pipe, '-|', $command or die "Could not open fh: $!";

while (<$pipe>) {
    print "Read: $_";
}

# avec IO::Pipe
use IO::Pipe;

my $pipe = IO::Pipe->new;
$pipe->reader( "$^X -V" );

while (<$pipe>) {
    print "Read: $_";
}

# Dans le sens de l'écriture

open my $pipe, '| $command' or die "Could not open fh: $!";
print $pipe "I can count to $_\n" for 1 .. 10;

# Avec IO::Pipe
```

```
use IO::Pipe;

my $pipe = IO::Pipe->new;
$pipe->writer( $command );

print $pipe "I can count to $_\n" for 1 .. 10;
```

Module *IO::Null*

```
use IO::Null;
```

my $null_fh = IO::Null->new;

- Module *IO::Interactive*

```
# Le livre n'indique pas d'importer 'interactive', mais c'est nécessaire.
use IO::Interactive qw(interactive);

# Les accolades sont obligatoires, sans quoi la valeur retournée par la sub
# interactive (par ex. "GLOB(0x56154b5b5b00)") est affichée : en l'absence
# d'accolades, print ne comprend pas que interactive est un handle de fichier.
print { interactive } "youpi\n";
```

- Un moyen rapide de capturer en vérifiant

```
while (<>) {
    unless (/^([^:]+):/) {
        warn "ignoring the line with missing name: $_";
        next;
    }
    my $name = lc $1;

    ...

}
```

- Smart match pour tester plusieurs regex à la fois

```
# To avoid a warning about feature being experimental
no warnings 'experimental';

# To use 'any', in order to make smart match clearer
use Smart::Match qw(any);

my @patterns = (
    qr/(?:Machin)?Bidule/,
    qr/truc/,
    qr/chou|ette/,
);
```

```perl
my $str = '...';

say "Match!" if $str ~~ @patterns;
# Or, along with
#   use Smart::Match qw(any);
say "Match!" if $str ~~ any(@patterns);
```

- Position des groupes de capture

```perl
# @- pour le début, @+ pour la fin
# $-[0] est pour le match entier, $+[0] également
if ($str =~ /$regex/) {
    print "Trouvé depuis la position $-[0] à la position $+[0]\n";
}
```

- Expressions régulières préconçues

```perl
use Regexp::Common qw(URI);
while (<>) {
    print "Ok\n" if /$RE{URI}{HTTP}/;
}

# Voir aussi :
print if /$RE{num}{int}/;
print if /$RE{num}{int}{ -base => 16 }/;
print $1 if /$RE{num}{int}{ -base => 16 }{ -keep }/;
print $1 if /$RE{ -base => 16 }{num}{ -keep }{int}/; # Fonctionne aussi !
```

- Assemblage d'expressions régulières

```perl
use Regexp::Assemble;

my $regex = Regexp::Assemble->new;
for ('ici', 'illico', 'là-bas', 'il fait beau', 'là-dessus') {
    $regex->add( "\Q$_" );
}
say $regex->re;

# Remarque : à partir de v5.10, cette optimisation est faite automatiquement.
```

- Lire des expressions régulières depuis un fichier et les tester

```perl
open my $fh, '<', 'patterns.txt';

my @patterns;

while (<$fh>) {
    chomp;
    my $pattern = eval { qr/$_/ }
        or do { warn "Invalid pattern: $@"; next; }
```

```
    push @patterns, $pattern;
}
while (<>) {
    foreach my $pattern ( @patterns ) {
        print "Match of [$pattern] at line $.: $_" if /$pattern/;
    }
}
```

- Tri stable

```
use sort "stable"; # Pour avoir des tris stables
```

- Mesurer des temps d'exécution

```
use Benchmark qw(:all);

# Si $count est négatif, il s'agit d'un nombre de secondes
timethese( $count, { key1 => sub1, key2 => sub2, ... } );
cmpthese( $count, { key1 => sub1, key2 => sub2, ... } );
```

- Transformation de chaînes pour la comparaison

```
$string =~ tr/a-z//cd      # Supprime tout caractère qui n'est pas une lettre
$string =~ s/\P{Letter}//g # Supprime tout caractère qui n'est pas une lettre
                           # Mieux que tr, car critère compatible Unicode.
$string =~ tr/A-Z/a-z/;    # Force à passer en minuscule
$string = fc ( $string )   # À partir de 5.16 ; plus propre que lc ou uc

# D'après le livre, la bonne méthode est de passer par
#   Unicode::Collate
# (non décrit par le livre)
```

- Le package courant est lexically-scoped

```
package Navigation;

{
    package main;
    sub mafonction1 { # main::mafonction1
        ...
    }
}

sub mafonction2 { # Navigation::mafonction2
    ...
}
```

- Variables toujours dans le package *main*

Les variables suivantes sont toujours dans le package *main* : `ARGV`, `ARGVOUT`, `ENV`, `INC`, `SIG`, `STDERR`, `STDIN`, et `STDOUT`.

`ARGVOUT` est le handle du fichier de sortie en cas d'exécution avec l'option `-i` (édition sur place).

- Les différentes syntaxes de *package*

```
package Navigation {
    ...
}
# est la même chose que
{
    package Navigation;
    ...
}

# On peut aussi donner un numéro de version
use 5.012;
package Navigation 0.01;
package DropAnchor 1.23 {
    ...
}
```

- Les deux systèmes de build

```
use ExtUtils::Makemaker; # Based on Makefile.PL
# versus
use Module::Build;       # Based on Build.PL
```

- Module::Starter

Installer Module::Starter (sous Ubuntu) avec

```
sudo apt install libmodule-starter-perl
```

Ensuite exécuter (exemple avec Text::AutoCSV)

```
module-starter --module=Text::AutoCSV --author="Sébastien Millet"
--email=milletseb@laposte.net --verbose
```

Le mieux est d'avoir un fichier de configuration pour Module::Starter.

Exemple, fichier `~/.module-starter/config` :

```
author: Sébastien Millet
email: milletseb@laposte.net
builder: Module::Build
verbose: 1
```

Ce qui permet d'exécuter simplement :

```
module-starter --module=Text::AutoCSV
```

Pour détecter les dépendances : `scandeps.pl`

- Instructions pour `Module::Build`

```
module-starter --mb --module="MyModule"
# Build.PL peut être modifié en fonction des besoins...
perl Build.PL
./Build
./Build test
./Build disttest
./Build dist
```

- Instructions pour `ExtUtils::Makemaker`

```
module-starter --builder="ExtUtils::Makemaker" --module="MyModule"
# Makefile.PL peut être modifié en fonction des besoins...
perl Makefile.PL
make
make test
make disttest
make dist
```

- Au sujet de `@ISA`

```
# Pour qu'une classe hérite d'une classe parente, les trois possibilités
# ci-dessous sont équivalentes.

# 1 (without using the parent class)
package Cow;
use vars qw(@ISA);
@ISA = qw(Animal);

# 2 (without using the parent class)
package Cow;
our @ISA = qw(Animal);

# 3
package Cow;
use Animal;
our @ISA = qw(Animal);

# 4
use v5.10.1;
package Cow;
use parent qw(Animal);
```

- Redéfinition (surcharge ?) de méthodes

```
# PAS BON !
# Pas de recherche de la bonne méthode dans la hiérarchie d'héritage.
sub speak {
    my $class = shift;
    Animal::speak($class);
```

```
    ...
}

# OK !
# Mais, il y a mieux - d'ailleurs, en cas d'héritage multiple, il faudrait
# savoir où chercher la méthode qui nous intéresse.
sub speak {
    my $class = shift;
    $class->Animal::speak;
    ...
}

# LA BONNE SOLUTION
sub speak {
    my $class = shift;
    $class->SUPER::speak;
    ...
}
```

- `TODO` dans `Tests::More`

Un bloc de code avec le label `TODO` pourra être rapporté comme non ok, mais il ne sera pas comptabilisé comme une erreur et n'empêchera pas les tests d'être réussis. Cela permet de différer une correction.

- Mesurer la couverture des tests

Avec Module::Build :

```
./Build testcover
```

Avec ExtUtils::Makemaker (non testé) :

```
HARNESS_PERL_SWITCHES=-MDevel::Cover make test
```

Les résultats des tests sont affichés et les détails sont écrits dans le répertoire `cover_db`.

- Méthodes de classe versus méthodes d'instance

Pour restreindre une méthode à être exécutée sur une instance :

```
# Uniquement objet
sub met_only_for_obj {
    ref(my $self = shift) or croak "...";
}

# Uniquement classe
sub met_only_for_class {
    ref(my $class = shift) and croak "...";
}
```

- Classe `UNIVERSAL`

Tous les objets dérivent de cette classe. Ainsi une méthode telle que

```
sub UNIVERSAL::youpi {
    say shift, " est content";
}
```

fournira la méthode **youpi** à tout objet quel qu'il soit.

- `DOES` et `can`

Ces méthodes qui sont toujours disponibles (elles font partie de la classe **UNIVERSAL**) permettent de vérifier le type d'objet (DOES) ou la disponibilité d'une méthode (can). can renvoie une référence sur la méthode quand elle existe, ce qui permet d'invoquer des fonctions dont le nom est connu dynamiquement :

```
my $metref = $myobj->can($metname);
$myobj->$metref(...) if $metref;
```

- `AUTOLOAD`

Si une méthode n'est pas trouvée, c'est en dernier ressort **AUTOLOAD** qui est appelé, ce qui permet de créer des méthodes dynamiquement.

- `EXPORT_TAGS`

Permet de grouper les imports avec des labels tels que `:all`, `:truc`, etc.

- Destruction d'objet

La méthode `DESTROY` est appelée par Perl lors de la destruction d'un objet. Perl l'exécute comme toute autre méthode en prenant en compte l'héritage.

- Tests

Il est possible de sauter des tests dynamiquement, avec le label `SKIP` et l'instruction `skip`.

Il est possible de grouper les tests avec `subtest`.

- `Test::LongString`

Ce module permet de n'afficher que ce qui diffère entre une chaîne attendue et ce qui a été obtenu, pratique si le teste porte sur de grandes quantités.

- `Test::File`

Fournit `file_exists_ok` et `file_not_exists_ok`

- `Test::Output`

Permet de vérifier ce qu'affichent des fonctions sur la sortie standard (ou sur l'erreur standard), avec `stdout_is` et `stderr_is`, `stdout_like`, `stderr_like`.

- `Test::Warn`

Permet de etster les warnings produits par l'exécution.

- `Test::NoWarnings`

Ce teste ne passe pas si un warning se produit.

- `Test::Builder`

Pour créer des fonctions de test.